SGI® UV2

A Fused Computation and Data Analysis Machine

Greg Thorson
Chief Engineer
SGI
Chippewa Falls, Wisconsin, USA
gmt@sgi.com

Michael Woodacre
Chief Engineer
SGI
Chippenham, Wiltshire, UK
woodacre@sgi.com

Abstract—**UV2 is SGI's second generation data fusion system. UV2 was designed to meet the latest challenges facing users in computation and data analysis. Its unique ability to perform both functions on a single platform enables efficient, easy to manage workflows.**

This platform has a hybrid infrastructure, leveraging the latest Intel® EP processors providing industry leading computational power. Due to its high bandwidth, extremely low latency NUMALink®6 (NL6) interconnect, plus vectorized synchronization and data movement, UV2 provides industry leading data intensive capability. It supports a single operating system (OS) image up to 64TB and 4K threads. Multiple OS images can be deployed on a single NL6 fabric, which has a single flat address space up to 8PB and 256K threads. These capabilities allow for extreme performance on a broad range of programming models and languages including OpenMP[1], MPI, UPC[2], CAF[3] and SHMEM.

The architecture, implementation and performance of UV2 are detailed.

I. INTRODUCTION

For more than a decade, high performance computer system deployments have been driven by the compelling economics of clusters, with message passing interconnects (MPI). MPI has become the dominant programming tool, and increasing efforts have been made to localize computation. This takes advantage of the local computing performance of the x86 processors and minimizes the communications between nodes which run at a fraction of the bandwidth compared to on-node memory access. The end of Dennard scaling[4], while Moore's Law continues, has led to microprocessors moving to the world of coherent shared memory multi-core processors. This has put pressure on a pure MPI programming direction, as the communication costs between cores have become non-uniform across cores on a die, cores within a node, and cores in a remote node. This has led to increasing interest in hybrid programming models (such as OpenMP-MPI) as well as new types of programming languages (PGAS), which can take full advantage of both coherent shared memory (CSM) and global shared memory (GSM).

Along with this fundamental technology shift, the emergence of the world of 'big data' driven by instruments, sensors, web content, digital transactions, social media, etc. has created huge volumes of digital data. This has, in turn, changed the way knowledge is gained.

Increasingly, knowledge discovery is driven by data analysis, coupled with computation, rather than relying primarily on computational simulations. In recent years, these data lead computation models have been driven in the internet world (i.e. map-reduce style programming). They are now starting to impact the technical computing arena.

This seismic shift in the computing industry demands that platform architects optimize their high performance computing (HPC) infrastructure around a new, broader, dynamic range of high performance technical computing. The UV2 data fusion architecture meets this need. It enables efficient implementations of these advanced algorithms, with support for critical HPC programming models, while seamlessly supporting today's standard applications. Different communication patterns can dynamically select the appropriate communication mechanism – leveraging coherent shared memory for local communication, and global shared memory for data transfers across very large systems. The ability of the processor to directly issue coherent transactions, and also initiate explicit global shared memory communication, provides the ideal base for programming models to dynamically select the most appropriate transport. This particular architecture provides closely coupled explicit communication and cache coherence on a common interconnect fabric These capabilities make UV2 equally suitable for leading edge data analysis including in-memory database and graph analysis applications, as well as traditional HPC applications.

UV2 supports the Intel® x86 'SandyBridge' EP (Efficient Performance) processors. This support enables UV2 to run a complete range of off-the-shelf software or custom-developed software, which is especially important when combining computation and data analysis onto a single platform. It provides the perfect sandbox for research that uses a variety of programming models and existing software.

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

All figures used with permission.
© 2012 Silicon Graphics International Corp.

Looking to the future, as processing capability continues to grow, the operational cost of large scale systems becomes an increasing part of overall total cost of ownership. It also becomes increasingly costly to move data into or within a system. Bringing together computation and data analysis, along with integrated storage, on one platform will enable a new generation of productive application workflows without the costly data movement overhead. Real time computation steering based on simultaneous computation and data analysis promises for the first time to enable interactive knowledge building.

II. System Architecture

UV2 uses Intel® processors optimized for HPC, coupled at the system level with low latency, high bandwidth interconnect, as well as highly scalable memory and I/O capabilities. These are requirements in its role of fusing HPC and big data applications into one easy-to-use platform. It is especially adept at combining complicated workflows inside a single system that is easy to install and administer.

The first UV2 product developed by SGI is the UV 2000 system as shown in Figure 1.

Figure 1. UV 2000 Configuration of 4K threads and up to 64TB of memory

This system is built from nodes that are connected in the NL6 network. It can be configured in a hierarchical structure of tightly coupled fat-nodes, where a fat-node is simply a hardware cache-coherent collection of physical nodes. Within and between fat-nodes, UV2 supports a flat address space (8PB) with interesting properties as described below.

This results in a unified fabric of cache coherent shared memory, global shared memory and message passing. The idea is to avoid the complexity and performance impact typically required when switching between data movement in cache coherent nodes and across message passing interfaces over cluster interconnects. For example, SGI's MPI library, MPT, automatically chooses the best communication mechanisms available in a given circumstance, and completely hides the selection under the application program interface (API) of the MPI library.

A. Cache Coherence

UV2 cache coherence is a hardware implementation based upon a directory-based scheme. This type of coherence was first implemented by SGI in the Origin® 2000 product[5]. This was chosen since software cache coherence schemes have good performance on only a limited set of applications, typically when the application has been decomposed to minimize communication due to the software overheads involved. The data fusion ASICs of the UV2 extend the cache coherent scalability from four or eight sockets of the native Intel® Quick Path Interconnect[6] (QPI), up to 512 processor sockets in a single hardware cache coherent domain. The system supports ~62TB of cache coherent shared memory. This is less than the 64TB address space the processor supports because of the fact that I/O and directory consume some of the space. The directory storage typically consumes approximately 3% of the memory space. It is important to note that the UV2 system provides a strong balance of memory capacity coupled with memory bandwidth. As memory capacity is added to the system, by adding nodes within a coherent shared memory region, memory bandwidth is also increased rather than just adding extreme memory capacity behind a fixed bandwidth channel in the system.

B. Explicit Communication Model

In addition to the optimizations for cache coherence, UV2 provides unparalleled explicit communication reminiscent of the Cray Research T3E[7]. Besides MPI, it has highly tuned hardware and software for partitioned global address space (PGAS) models such as support for SHMEM and Unified Parallel C (UPC). Besides the flat address space, the system supports GETs, PUTs and atomic memory operations (AMOs) across that 8PB address space. These operations are coherent, but not cached. That is, GETs will typically retrieve a datum directly from memory, but will get it from a processor cache if it is modified within that cache. Likewise, a PUT writes to memory and invalidates any caches that contain previous copies of data from that address. This means that the hardware prevents a thread from spinning on a stale copy of the data. Note that AMOs behave similarly as they are largely hybrids between PUTs and GETs. The system also supports vector transfers for PUTs, GETs and AMOs in order to increase performance.

When working within an OS image, the explicit communication and the coherent memory accesses can use the same virtual addresses.

C. Partitioning

Even though UV2 supports an extremely large cache coherence domain, it can also be partitioned into a number of smaller OS instances of varying sizes, as small as two processor sockets. It can also collect numerous partitions into

a single flat address space to 8PB. The operating systems of different partitions communicate with each other, if desired, to expose sections of their memory for access through this global shared address space. Partitions can also be used to support different operating environments within the system. These can be booted independently of each other to allow for both system maintenance and resizing of OS images.

D. Packaging and Topology

UV2 is packaged in the UV 2000 product as an 8 blade, 16 processor socket, enclosure as shown in Figure 2. Four of these enclosures can be co-located in a standard 19" rack.

Figure 2. UV 2000 enclosure with 8 blades

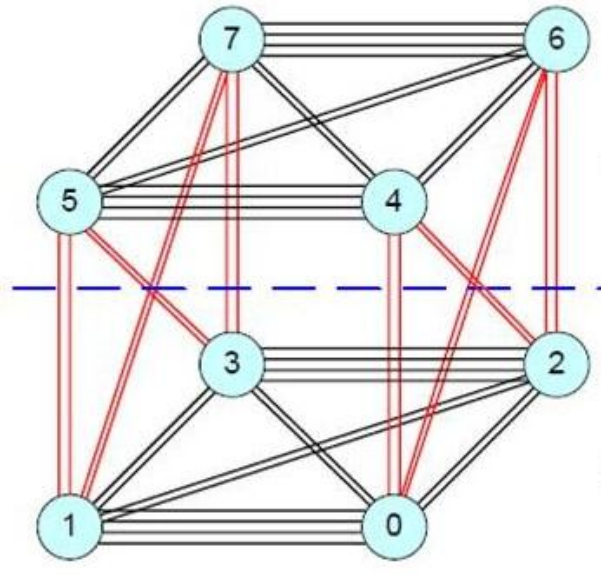

Figure 3. 16 socket topology

The topology, shown in Figure 3, is integrated in a backplane at the enclosure level. This is flexible above that by using cables, as four ports are free on each node to allow for this flexibility. For example, the enclosures can be directly cabled to each other as in Figure 4, or up to 32 external NL6 routers can be added to a 128 socket unit. Four such routers are shown connected to one of eight vertices in Figure 5, but in many configurations all eight vertices would be connected to their own set giving 32 independent router planes. This particular arrangement is a building block configuration. Using the 8 spare ports per router at the top of the picture, these building blocks can be connected in a variety of all-to-all, fat-tree or torus arrangements. These upper level topologies have varying latencies, bisection bandwidths, and point-to-point bandwidths.

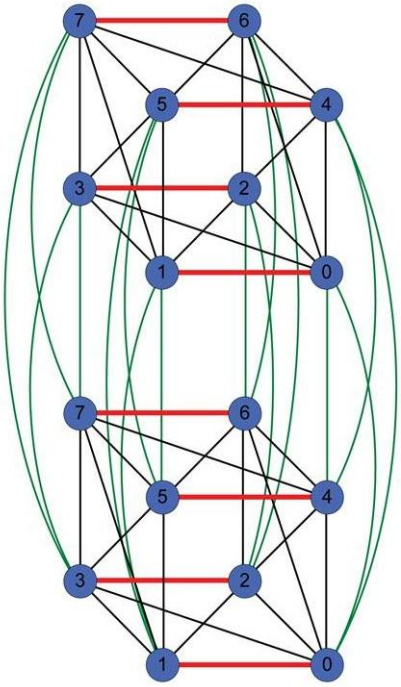

Figure 4. Direct cabled 32 socket topology

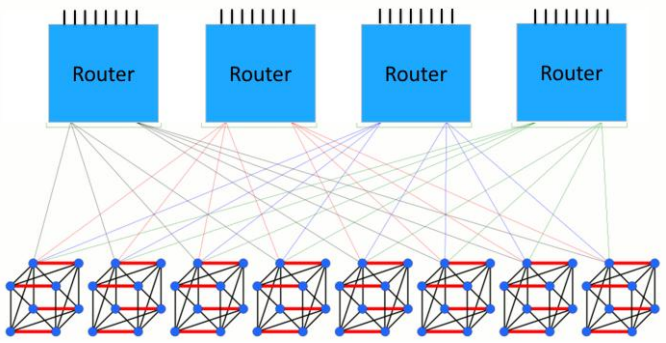

Figure 5. Topology with external routers

E. I/O

When building a system intended for dealing with large datasets, it is critical to provide balanced I/O capability for moving data in and out of the system, and to support persistent storage. The latest Intel® processors integrate PCIe I/O interfaces directly onto the socket. The UV 2000 packaging leverages this capability by providing PCIe slots on each blade that can be used to provide external I/O fabric connections to high speed networking and storage devices; for example, to Ethernet, InfiniBand, or Fibre Channel. UV 2000 also supports the use of FLASH storage cards directly on the node using this PCIe capability of the processor socket. FLASH storage offers lower power and higher performance compared to traditional disk-based storage.

The data fusion ASIC detects I/O data streaming across NL6 and uses a pre-fetch mechanism to increase I/O performance across the interconnection network.

The ability to connect substantial I/O on all the nodes within a single system enables extreme ingest and egress of data directly to and from system memory. Each of the UV 2000 blades has access to 16 PCIe Gen3 lanes per processor socket. This provides a peak bandwidth of 16GB/s per processor, combining to a current limit of 4TB/s peak I/O bandwidth within a single operating system or 256TB/s across a fully configured 256 rack system.

F. Global Clock Synchronization

For a number of applications (e.g. databases), ensuring that all threads within the application have a synchronized time is of great importance. Each x86 processor has its own time stamp counter (TSC). The TSC counts the number of cycles since reset, and the RDTSC instruction can be used to read the current value. The UV2 system supports the option to use its management network to distribute a global time reference to the local clock generators. This ensures that all individual processor TSCs remain in sync, regardless of the fact that the system contains many processors on many different blades.

III. Node Architecture

A node consists of two Intel® 'Sandy Bridge' processor sockets, with up to 8 cores each, connected to the data fusion ASIC. Figure 6. highlights the blade-based packaging of a node. This particular form factor optimizes for: I/O support, high-speed signaling, backplane connection count, and flexible cable connection count. As shown in the figure, 12 NL6 ports connect to the backplane and the other 4 are available for connection to cables out the front.

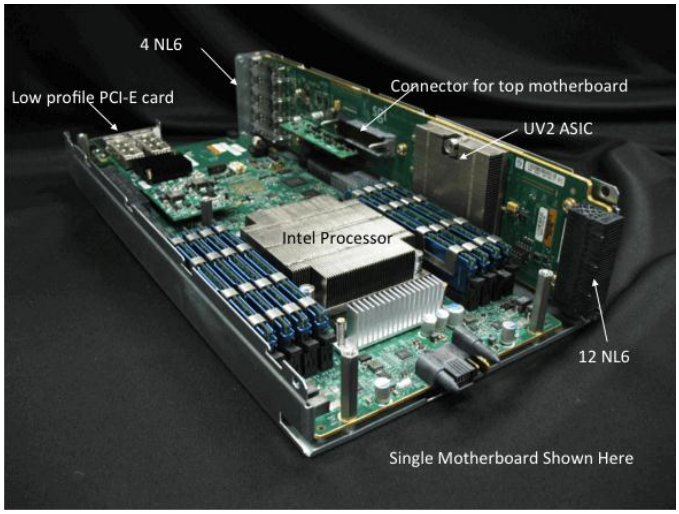

Figure 6. UV 2000 blade (top processor board removed)

IV. ASIC Architecture

The data fusion ASIC is implemented on a 18mm x 18 mm die with 413 million transistors. It contains, among other features, a processor socket interface, a coherent request engine, a coherent home engine, an explicit communication engine, and an integrated router, as seen in Figure 7. In fact, there are two full sets of these resources, as one set is dedicated to each processor socket. The exception is that the two halves of the router are shared by both processor sockets to provide more bandwidth and/or connectivity. Note that the communication engine has its functionality split between source and destination components. These are the Global Reference Unit (GRU) and the Active Memory Unit (AMU), respectively.

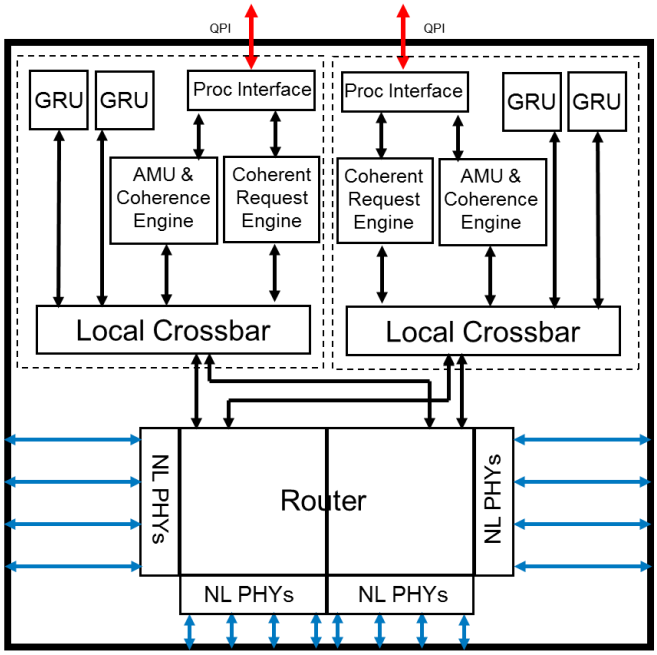

Figure 7. Data fusion ASIC block diagram

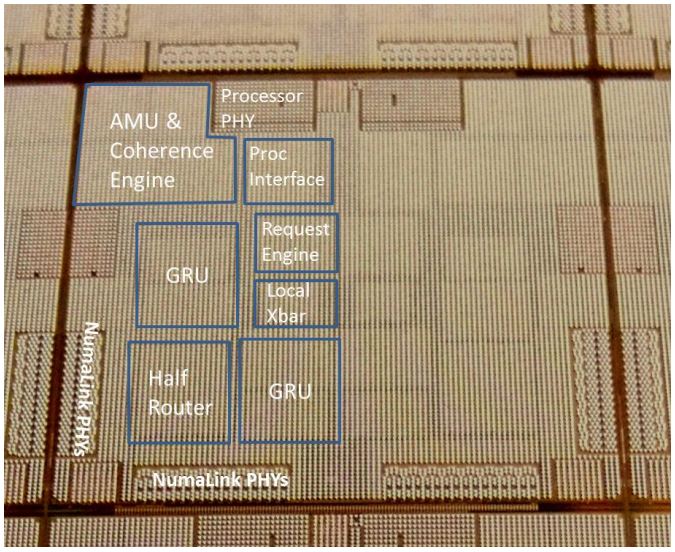

Figure 8. Annotated photo of data fusion ASIC

A. *Processor Socket Interface*

The processor socket interface is the first stage of torque conversion between Intel® QPI and SGI's highly scalable NL6. This interface consists of the processor physical layer (PHY), a fast snoop responder, as well as routing functions between the socket and the various coherence engines. Note that the data fusion ASIC is optimized so it does not interfere with the processor socket's local memory performance, unless it is actually processing remotely touched data. For traffic heading towards the socket from the coherence engines, the processor interface is responsible for reformatting into QPI packets.

B. *Coherent Request Engine*

Remote memory requests are passed from the processor interface to the coherent request engine. This block assigns resources that store information required for conversion back to QPI protocol on return. Conflicts with probes from remote coherent home engines, end-to-end retries, and errors are also handled by the coherent request engine. The pre-fetch capability mentioned earlier is provided by this block.

C. *Coherent Home Engine*

The coherent home engine provides such functions as multi-node coherence tracking and multicasting of probes and invalidations. Coherence is assured via a directory structure that is maintained by the coherent home engine. Directory entries contain sharing vectors or pointers to deal with shared or exclusive data, respectively. When data is to be invalidated from remote caches, the coherent home hardware utilizes its facility for parallel multicasting of the invalidation messages. In fact, this multicasting system provides for combining of multicast responses to further reduce any impacts on system bandwidth and latency. Although the full directory is stored in socket DRAM, this block has a large 64-way directory cache to provide higher performance on reused addresses or patterns with good strides.

D. *Global Reference Unit*

A second block can generate requests above and beyond those handled by the coherent request engine. The GRU supports a number of key features that enable the UV2 with Intel® Xeon® processor sockets to scale to more than 128K processor cores and 8PB of memory.

The GRU has control registers and a large set of vector registers. These registers are memory mapped at the user level and thus are read and written with processor loads and stores, respectively. The registers contain GET/PUT data, user level virtual addresses, and control; thus, kernel calls are only required for allocation and maintenance purposes. Access to a given memory address is provided by an OS maintained translation look-aside buffer (TLB) in the GRU. This prevents a user's access to unauthorized resources. To perform vector or scalar GETs, the addresses are stored in GRU registers and the operations are triggered by writes to the control registers. The resultant data is then read from the appropriate GRU register. PUTs behave similarly except that their data are written to the registers before the PUT is triggered by the control register write.

The Intel® Xeon® processors used in this system have a 46-bit physical address space and a 48-bit virtual space. The GRU's TLB extends this physical address space to 53 bits and the virtual space to 60 bits. Within a single OS region, the user can apply the same virtual addresses to both coherent socket accesses and GRU generated accesses. When communicating between multiple OS regions, an extended address must be used. This extended address is a single 60-bit value and does not require any sort of segmentation.

When dealing with large memory systems, and in particular for applications that require data access with little locality, a lot of pressure is put on the address translation mechanisms in the system. The x86 processors typically use four kilobyte pages, but do support larger page sizes such as two megabytes. The use of these huge pages significantly reduces TLB pressure and can lead to large performance improvements for locality challenged applications. In addition to the standard x86 page sizes, the GRU TLB supports super-pages that can be multiple gigabytes in size. All factors of four above 1GB, up to 1TB, page sizes are supported. The GRU's TLB also provides shoot-down capabilities to increase the speed with which a large memory job, such as a very large in-memory database, can be started or terminated.

Within memory systems of tens of terabytes, page initialization can be a major performance bottleneck. UV2 enhancements allow pages of memory to be initialized in a distributed manner with little involvement from the x86 processors. By off-loading memory initialization from x86 processors onto the distributed network of data fusion ASICs, initialization times are dramatically reduced.

Block Copy (BCOPY) operations provide asynchronous

mechanisms to copy data from one memory location to another. This is commonly found when replicating read-only data structures within threaded applications to avoid contention, or in message passing environments to send an actual message from one blade to another. SGI shared memory systems offer very low message passing latency, partially because BCOPY is used to move aligned data structures that represent the majority of bytes transferred. The BCOPY functionality in the data fusion ASIC extends this capability. It supports offloading the copying of unaligned portions of data transfers from the x86 processors, and making BCOPY directly accessible from user space. This significantly reduces the message passing overhead. BCOPY is used to transmit the actual data from one blade to another without x86 processor involvement.

Vector memory operations such as fixed stride and list-driven scatter/gather memory transfer operations are processed directly by the GRU. These operations allow random addresses to be accessed without the need to carry whole cache lines around the interconnection network, improving effective bandwidth and reducing latency over the NL6 interconnect. Vector memory operations assemble a vector of related, but not necessarily contiguous, data elements into contiguous locations within the global shared memory address space, or distribute values from contiguous locations to specified memory locations anywhere within the global shared memory address space. The staging areas for vector data and addresses are accessed as a large set of vector registers that are assigned on a per context/thread basis. Vector scatter/gather operations have a long history of increasing achievable performance by optimizing memory transfer patterns, and scatter/gather operations implemented in the GRU make these optimizations available to Intel® Xeon® processors, creating cache friendly data structures that can be processed with little or no x86 processor stalls.

Vector AMOs are supported to provide very fast global updates (i.e. GUPS). These can stride or scatter through an indirection vector. The vector AMOs are treated by the GRU as scatters and do not return a value.

The GRU has a cache that allows for globally accessed variables, such as barrier counts and contended locks, to spin locally on a shared copy of an AMO variable, reducing hot spots in the home memory. Contended locks and reduction variables can see up to two orders of magnitude acceleration in access times, which becomes critical for very large problems spanning large numbers of threads.

Support for MPI, SHMEM, and UPC are provided to make use of the GRU. An API is also supported for direct access to the GRU. This is to allow performance optimization as well as providing an interesting mechanism for PGAS research.

E. Active Memory Unit

The Active Memory Unit (AMU) provides numerous functions, including: home-based AMOs, multicasting of modifications to cached atomic memory operations (AMOs), and message queue reception. All of these functions are implemented in user memory and require no intervention or communication with the kernel. The AMU's multicast and coherence functions piggyback on the coherent home engine's directory functionality.

Five types of atomic memory operations (AMOs) are enabled by the AMU.

The first type is compatible with the AMOs implemented by Intel® Xeon® processors. A user can simultaneously use the Intel® AMOs and the SGI AMOs on the same memory addresses without loss of coherence. Uncontended variables within a coherence domain are best accessed via AMOs on the processor socket and reused within its cache. Contended variables are best accessed via the AMUs AMOs. The contended variables data can then be reused within the data fusion ASIC's AMO cache, reducing coherence overhead. To provide this improved access, the home node's data fusion ASIC takes ownership of the data from the attached processor socket and stores that data in its AMO cache. The AMU's AMO cache provides low latency, high repeat rate access to active synchronization variables across the NL6 interconnect.

A second type of AMO at the AMU behaves similar to the type one operations, but these are additional functions beyond what are supported by the socket.

A third type of AMO is cached in a shared state in the GRU cache at the requesting nodes. When a datum changes state, the AMU sends modifications or invalidates to those caches to maintain coherence. This multicasting greatly improves the latency and repeat rate of GRU cache based AMOs. This is especially useful for barriers, collectives and contended locks.

A fourth type of AMO is used for global updates (i.e. GUPS). These atomically modify the memory, but do not return a value to the requester. These particular AMOs can be vectorized as described in the GRU section above.

The final type of AMO is used to implement message queues. Just as with other AMOs, message queues are implemented in standard user memory so a program can set up as many queues as desired, as there are no specialized resources to be reserved. 64 byte and 128 byte messages are sent by the GRU directly to the address of a queue pointer, and the hardware takes care of storing at the appropriate offset from that pointer's address, updating the queue pointer with an AMO and protecting against queue overflows via negative acknowledgements.

F. Router

The UV2 interconnect is implemented using the SGI NL6 hardware stack. The physical connections are compatible with InfiniBand x4 cables. NL6 supports both passive copper and active optical cables. The physical, link and protocol layers are NL6-specific. The physical layer operates each signal lane at 12.5 GT/s. The link layer supports link-level retry for

resilience. Retry logic and virtual channel queuing support up to 30m of optical cable at full bandwidth. The copper and optical cable options combined with the 16 port routing capability opens up topology options including all-to-all, fat-tree or torus. The protocol layer is optimized for cache line-sized (64 bytes) payloads, as well as double-word (64 bit) payloads, in contrast to standard network protocols that are optimized for larger message payloads. The performance of small messages is increasingly important for parallel efficiency.

A router is integrated in the data fusion ASIC. 16 ports of this router are exposed to the outside. This supports low latency and high bandwidth operation even without external routers. External NL6 routers are added in larger configurations to meet the latency and bandwidth requirements of such systems.

V. System Operating Environment

UV2 is designed to run standard, off-the-shelf, x86 operating systems. Initially the system is running Red Hat® Enterprise Linux® (RHEL 6.3) and SUSE® Linux Enterprise Server (SLES11 SP2). SGI has worked with the open source Linux community, contributing necessary kernel elements to scale the operating system to manage 4,096 threads and 64 terabytes of memory. Typically a core is configured to support one or two threads, so this translates to a maximum of 256 or 512 processor sockets per operating system image. Having a single operating system span this large resource greatly eases system management, by reducing the number of OS images by two orders of magnitude compared with a cluster.

On top of the operating system, SGI provides software packages for software development and performance acceleration (SGI Development Suite and SGI Performance Suite, respectively). A key piece of the performance acceleration capability is the message passing toolkit (MPT), which leverages the hardware capabilities of the underlying platform, wrapping them up for ease of use by the MPI communications library, UPC compiler and SHMEM communications library.

The Linux buffer cache is a mechanism for dynamically caching file contents in memory. With the size of memory available in a UV2 system, this mechanism often provides significant acceleration of workflows that share file data, without the need for any modification of the applications.

VI. Communications Engine API

The low level features of the UV2 communication engine within the data fusion ASIC are contained within an API that can be used to implement higher level communication models. Full documentation of the communication engine interface can be found on the SGI web site[8] .

An example piece of code is shown in Figure 9 that demonstrates how to use the communication engine to move data around the system instead of using the processor to move the data.

```
#include <string.h>
#include <uv/gru/gru.h>
#include <uv/gru/gru_instructions.h>
#include <uv/gru/gru_alloc.h>
void *egru_memcpy(void *dst, const void *src, size_tnbytes)
{
        int tri, lines;
        gru_alloc_thdata_t thd;
        if (gru_temp_reserve_try(&thd) != 0)
                return memcpy(dst, src, nbytes);
        lines = thd.dseg_size / GRU_CACHE_LINE_BYTES;
        lines &= ~1;
        tri = gru_get_tri (thd.dsegp);
        gru_bcopy(thd.cbp, src, dst, tri, XTYPE_B, nbytes, lines, 0);
        gru_wait_abort(thd.cbp);
        gru_temp_release();
        return dst;
}
```

Figure 9. Example GRU library routine use

The routines supported are split into two categories — communication data transfer and communication completion (TABLE I.). Completion notification is independent of data transfer to maximize the potential for overlapping communications and computation.

Memory and Message Operations	gru_bcopy,gru_bstore, gru_ivload, gru_ivset, gru_ivstore, gru_vload, gru_vset, gru_vstore gru_gamir, gru_gamirr, gru_gamer, gru_gamerr gru_send_message
Completion Operations	gru_wait, gru_wait_abort gru_check_status

TABLE I. GRU API Routines

VII. Performance

This section examines the performance of the UV 2000 product, initially using micro benchmarks. The results were obtained on early engineering hardware, and it is expected that with additional tuning the final product performance will exceed these numbers. As an example, the GUPs peak performance is expected to increase by 10-20%. Unless otherwise stated, these are measured with 8 cores per socket.

The first micro benchmarks examine the basic bandwidth and latency characteristics of the system. The STREAM benchmark is the most widely used memory bandwidth benchmark in the industry. The STREAM benchmark used is implemented using OpenMP to distribute the code across the cores in the system. Figure 10. shows that the triad demonstrates linear scaling as the system size increases. Figure 11. shows the uncontended read latency of the system. This highlights that it is possible to read any location in the entire 64TB cache coherent shared memory address space in under one microsecond. The global shared memory latency reaching across a 1PB sized system is estimated at less than three microseconds.

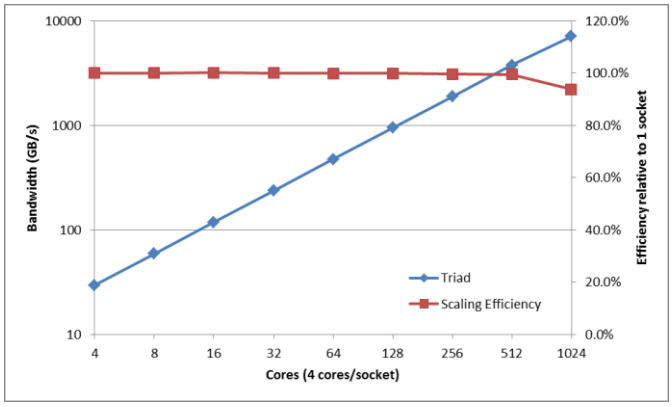

Figure 10. UV 2000 STREAM Performance

The maximum latency takes a step change going to 512 cores. This is due to the two additional NL6 hops required to support the larger configuration. When going from 2,048 to 4,096 cores (4 to 8 racks), one additional NL6 hop is added.

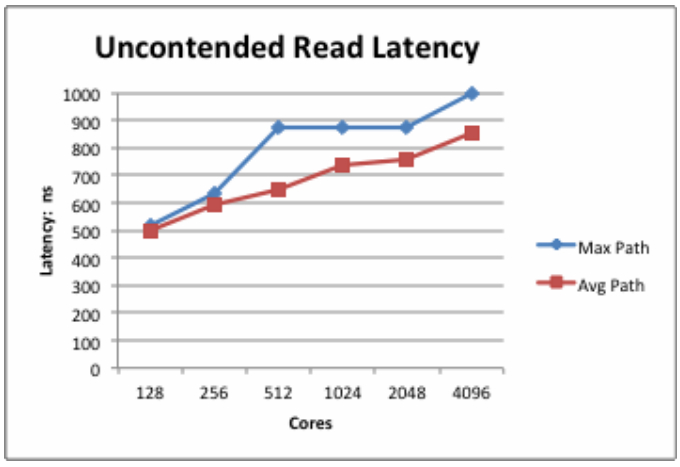

Figure 11. UV 2000 Read Latency (load-to-use time)

The next set of micro benchmarks demonstrates the effectiveness of the communication acceleration mechanisms supported by UV2. The MPI tests are all based on the Intel® MPI Benchmark (IMB) tests. Figure 12. shows the MPI barrier performance that can be achieved using the communication acceleration features of UV2. The offload capability enables the scaling of barrier performance measured out to 1,024 cores with under 3.5us of latency, and the underlying implementation is expected to continue with the low latency profile to very large system sizes.

Figure 13. shows the MPI collective performance of the all-reduce operation. This operation uses eight byte data for the operation. The profile of this graph again demonstrates the performance scaling capability of the communications engine. Note that the collectives do not have explicit hardware support, but are built with a software layer on top of the fast barrier mechanism.

Figure 14. shows the MPI broadcast performance. This is based on broadcasting an eight byte payload to all the cores in the system.

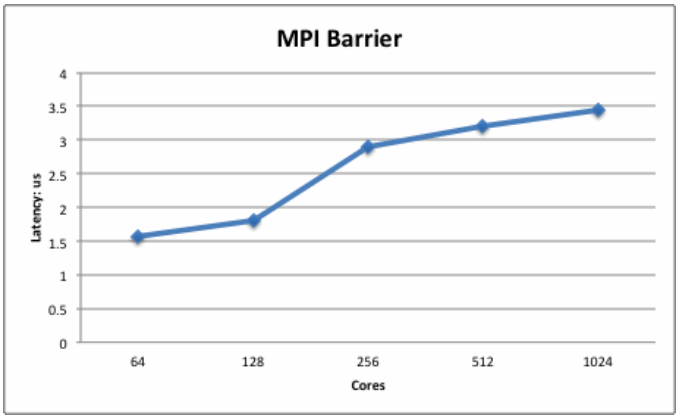

Figure 12. UV 2000 MPI Barrier Performance

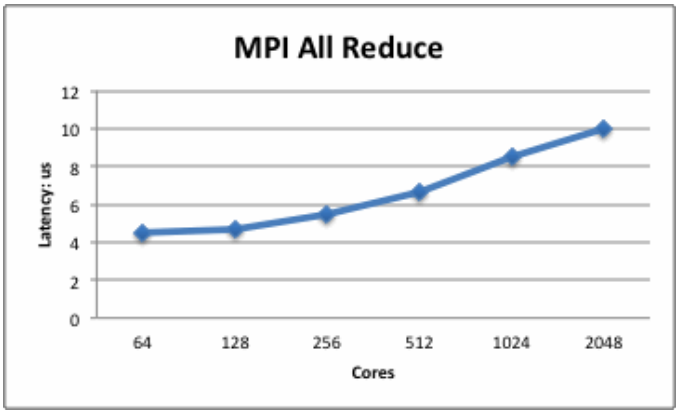

Figure 13. UV 2000 MPI AllReduce Performance

Global update performance is an important micro benchmark used to determine how well a system performs when data is updated randomly across the entire memory space. The version of GUPs used in this test operates on unsorted updates, without any look-ahead or bucketing of updates. Figure 15. shows the results obtained for GUPs performance using a range of techniques. The baseline performance used for comparison in this chart is the SGI UV 1000 system that was released in 2010. GUPs performance, by its global reach, places pressure on TLBs (both processor and communication engine). Using huge pages (HP) with the UV 2000 processor-driven GUPs outperforms UV 1000 communication engine

performance. Best performance is achieved when taking advantage of the super-page (SP) support in the communication engine (4GB pages, in this case), along with vector atomic memory updates. An interesting aspect of the implementation of GUPs using vector AMOs is that the 'error' rate generally associated with GUPs falls to zero, because the implementation performs the atomic updates at the destination end (home) of the data using the processor's coherent memory operation mechanism.

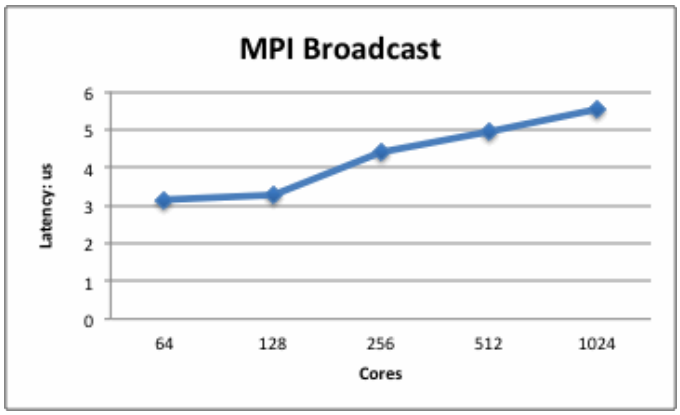

Figure 14. UV 2000 MPI Broadcast Performance

VIII. Conclusions

The SGI UV 2000 system provides the fusion of high performance computation with integrated data analysis capability. This has been achieved by using the latest Intel® x86 processor technology combined with an interconnection network focused on data acceleration. No compromise has been made in delivering the data intensive capability combined with outstanding single thread performance. This is a key attribute of the system that will meet the growing demands of hybrid applications and workflows, as well as the expanding range of workloads that modern HPC centers must support.

In addition to its leading performance capability for existing applications, this platform provides the capability for rapid innovation in new algorithms and workflows. For example, workflows will be needed to move towards real-time computation steering to maximize performance and minimize energy consumption.

IX. Future Directions

Computational processing and communication acceleration will most certainly move onto a single device. Lessons learned from workflow optimization on UV2 will provide insight into design tradeoffs required by these new devices and assure the continued advancement of large-scale data fusion computing.

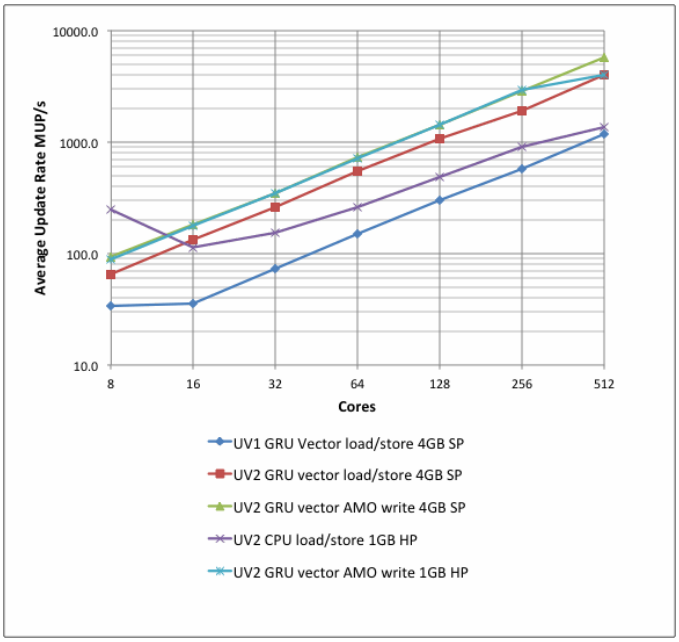

Figure 15. UV 2000 GUPS Performance

X. Acknowledgements

Development of the SGI UV 2000 system resulted from the dedication of a large team over a number of years, including engineers involved in ASIC design, board design, system power/package/cooling, system management, OS, performance tools and performance analysis. The authors thank the entire team for their hard work that enabled the results achieved. Special thanks go to John Baron and Karl Feind for providing the early performance results.

References

[1] Leonardo Dagum, Ramesh Menon, "OpenMP: An Industry Standard API for Shared Memory Programming", IEEE Computational Science and Engineering, January-March 1998

[2] Tarek El-Ghazawi, William Carlson, Thomas Sterling, Katherine Yellick. "UPC: Distributed Shared Memory Programming", ISBN: 978-0-471-22048-0

[3] Numrich and Reid (1998), Co-Array Fortran for Parallel Programming, ACM Fortran Forum, volume **17**, no 2, pp 1-31.

[4] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. IEEE Journal of Solid-State Circuits, 9, October 1974

[5] James Laudon and Daniel Lenoski. "The SGI Origin:a ccNUMA highly scalable server" In Proc. Of International Symposium on Computer Architecture (ISCA), pages 241-245, June 1997.

[6] Robert A. Maddox, Gurbir Singh, and Robert J. Safranek. "Architectural Insights to the Intel QuickPath Interconnect", Intel Press.

[7] Steven L. Scott, "Synchronization and Communication in the T3E Multiprocessor", In Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 26-36, October 1996.

[8] SGI UV GRU Programmers Guide, http://techpubs.sgi.com/, document number 007-5668-004